120

AF

Our Docket No.: 42P11329

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of:

Arch D. Robison	Examiner: Gross, Kenneth A.
Application No.: 09/552,292	**Art Group: 2122**
Filed: April 19, 2000	
For: Data-Flow Method for Optimizing Exception-Handling Instructions in Programs	

APPEAL BRIEF IN SUPPORT OF APPELLANT'S APPEAL TO THE BOARD OF PATENT APPEALS AND INTERFERENCES

Sir:

Appellant hereby submits this Brief in triplicate in support of its appeal from a final decision by the Examiner, mailed March 2, 2004, in the above-referenced Application. Appellant respectfully requests consideration of this appeal by the Board of Patent Appeals and Interferences for allowance of the above-captioned patent application.

An oral hearing is not desired.

TABLE OF CONTENTS

I. REAL PARTY IN INTEREST

The invention is assigned to Intel Corporation of 2200 Mission College Boulevard, Santa Clara, California 95052.

II. RELATED APPEALS AND INTERFERENCES

To the best of Appellant's knowledge, there are no appeals or interferences related to the present appeal that will directly affect, be directly affected by, or have a bearing on the Board's decision.

III. STATUS OF THE CLAIMS

Claims 1-6 and 10-17 are currently pending in the above-referenced application. In the Final Office Action mailed March 2, 2004, claims 1, 2, 10, 13 and 14 stand rejected under U.S.C. §103(a) as being unpatentable over "How Debuggers Work," by Jonathan B. Rosenberg, 1996 ("*Rosenberg*"), in view of Wallace et al. (U.S. Patent No. 6,018,799) ("*Wallace*") and further in view of *Lo* et al. (U.S. Patent No. 6,151,706) ("*Lo*"). In addition, claims 3 and 15 stand rejected under 35 U.S.C. §103(a) as being unpatentable over *Rosenberg* in view of *Wallace* and further in view of *Lo* and Gordon et al. (U.S. Patent No. 6,507,805) ("*Gordon*"). Further, claims 4, 5, 11, 12, 16 and 17 stand rejected under 35 U.S.C. §103(a) as being unpatentable over *Rosenberg* in view of *Lo* and further in view of Dunn et al. (U.S. Patent No. 6,247,172) ("*Dunn*"). Finally, Claim 6 stands rejected under 35 U.S.C. §103(a) as being unpatentable over *Rosenberg* in view of *Lo* and further in view of *Dunn* and *Gordon*.

IV. STATUS OF AMENDMENTS

In response to the Final Office Action mailed on March 2, 2004, rejecting claims 1-6 and 10-17 under 35 U.S.C. §103(a), Appellant filed an Amendment After Final pursuant to 37 C.F.R. § 1.116 on March 30, 2004. Subsequently, an Advisory Action was mailed on April 16, 2004. In response, Appellant filed a Notice of Appeal on June 2, 2004. A copy of all claims on appeal is attached hereto as an Appendix of Claims.

V. SUMMARY OF THE INVENTION

According to one embodiment, a method is described. The method (see e.g., Fig. 4) includes analyzing a program to determine the state of a data structure at selected program points (page 8, lines 14-18; Fig. 4step 400), partitioning the determined state at each program point into components that may each be set separately (page 8, lines 18-20; Fig. 4 step 401), determining operations to be inserted into the program in order to set each component of the state at each selected program point, wherein the operations assure that the data structure will be in an accurate state at the selected program points (page 8, lines 20-23; Fig. 4 step 402), and placing the operations to eliminate partial redundancies of the operations (page 8, lines 25-26; Fig. 4 steps 403-404).

In a further embodiment, another method is described. The method (see e.g., Fig. 4) includes analyzing a program to determine the state of an instance of a data structure at selected program points (page 8, lines 14-18; Fig. 4 step 400), partitioning said instance of said data structure into components (page 8, lines 18-20; Fig. 4 step 401), determining a set of one or more operations to be inserted into the program in order to set each component of the state at each selected program point, wherein the operations assure that the data structure will be in an accurate state at the selected program points (page 8, lines

20-23; Fig. 4 step 402), computing placement of the set of operations to eliminate partial redundancies (page 8, lines 25-26; Fig. 4 step 403), and inserting the set of operations at the program points according to the computed placement (page 8, lines 25-26; Fig. 4 step 404).

In yet another embodiment, a machine-readable medium is disclosed having a set of instructions which when executed by a set of one or more processors causes the set of processors to perform (see e.g., Fig. 4) analyzing a program to determine the state of an instance of a data structure at selected program points (page 8, lines 14-18; Fig. 4 step 400), partitioning said instance of said data structure into components (page 8, lines 18-20; Fig. 4 step 401), determining a set of one or more operations to be inserted into the program in order to set each component of the state at each selected program point, wherein the operations assure that the data structure will be in an accurate state at the selected program points (page 8, lines 20-23; Fig. 4 step 402), computing placement of the set of operations to eliminate partial redundancies (page 8, lines 25-26; Fig. 4 step 403), and inserting the set of operations at the program points according to the computed placement (page 8, lines 25-26; Fig. 4 step 404).

VI. ISSUES PRESENTED

Whether claims 1, 2, 10, 13 and 14 are patentable over *Rosenberg*, *Wallace*, and *Lo* in view of 35 U.S.C. §103(a);

Whether claims 3 and 15 are patentable over *Rosenberg*, *Wallace*, *Lo* and *Gordon* under U.S.C. §103(a);

Whether claims 5, 11, 12, 16 and 17 are patentable over *Rosenberg*, *Lo* and *Dunn* under U.S.C. §103(a); and

Whether claim 6 is patentable over *Rosenberg*, *Lo*, *Dunn* and *Gordon* under U.S.C. §103(a).

VII. GROUPING OF CLAIMS

The claims stand and fall together.

For the purposes of this appeal claims 1-6 and 10-17 stand or fall together as Group I.

VIII. ARGUMENT

1. Claim Group I

(A) THE PENDING CLAIMS WERE IMPROPERLY REJECTED UNDER 35 U.S.C. § 103(a) BECAUSE *ROSENBERG*, *WALLACE*, AND *LO* DO NOT DISCLOSE OR SUGGEST DETERMINING A SET OF ONE OR MORE OPERATIONS TO BE INSERTED INTO THE PROGRAM IN ORDER TO SET EACH COMPONENT OF THE STATE AT EACH SELECTED PROGRAM POINT, WHEREIN THE OPERATIONS ASSURE THAT THE DATA STRUCTURE WILL BE IN AN ACCURATE STATE AT THE SELECTED PROGRAM POINTS

Appellant respectfully submits that *Rosenberg*, *Wallace*, and *Lo* fail to disclose or suggest the claimed invention for the reasons set forth below.

Each claim in Claim Group I recites an element that is not disclosed in *Rosenberg*, *Wallace*, or *Lo*. For example, Appellant's claim 1 recites the following:

> For a computer-executable program that operates on a data structure, where the data structure must have a required state at selected program points, a method of transforming said program comprising:
> (A) analyzing the program to determine the state of said data structure at said selected program points;
> (B) partitioning said determined state at each said program point into components that may each be set separately;
> (C) determining operations to be inserted into the program in order to set each component of the state at each selected program point, wherein the operations assure that the data structure will be in an accurate state at the selected program points; and
> (D) placing said operations to eliminate partial redundancies of said operations.

Appellant's claim 10 recites:

> For a computer-executable program that operates on a data structure, where the data structure must have a required state at selected program points, a method of transforming said program comprising:

> (A) analyzing the program to determine the state of an instance of said data structure at said selected program points;
> (B) partitioning said instance of said data structure into components;
> (C) <u>determining a set of one or more operations to be inserted into the program in order to set each component of the state at each selected program point, wherein the operations assure that the data structure will be in an accurate state at the selected program points</u>;
> (D) computing placement of the set of operations to eliminate partial redundancies; and
> (E) inserting the set of operations at said program points according to the computed placement.

Appellant's claim 13 recites:

> A machine-readable medium having a set of instructions, which when executed by a set of one or more processors, causes said set of processors to perform operations comprising:
> (A) analyzing a program that operates on a data structure, which must have a required state at selected program points in the program, to determine the state of an instance of said data structure at said selected program points;
> (B) partitioning said instance of said data structure into components;
> (C) <u>determining a set of one or more operations to be inserted into the program in order to set each component of the state at each selected program point, wherein the operations assure that the data structure will be in an accurate state at the selected program points</u>;
> (D) computing placement of the set of operations to eliminate partial redundancies; and
> (E) inserting the set of operations at said program points according to the computed placement.

Rosenberg describes stack unwinding, which employs an algorithm for finding traces on a stack. Commands are implemented to unwind a stack to find a parent procedure's frame pointer and return address. See *Rosenberg* at page 136, lines 23-30. The algorithm for unwinding a traditional stack involves pushing return addresses onto the stack. A procedure call pushes a return address onto the stack and a child procedure pushes the parent's frame pointer address onto the stack. See *Rosenberg* at page 137, lines 23-29.

Lo discloses a method a system and method for extending sparse partial redundancy elimination (PRE) to support speculative code motion within an optimizing compiler. See *Lo* at col. 3, ll. 23-25.

Wallace discloses enabling a compiler to generate efficient code to access stack registers on a register stack. See *Wallace* at Abstract. In particular, a pseudo-register mapping process for mapping pseudo-registers to stack registers within the register stack is disclosed. The process includes an `iterate each instruction` procedure, which iterates each instruction in a basic block. Each instruction iterated by the `iterate each instruction` procedure is checked to determine whether the instruction accesses a pseudo-register at a `floating point register instruction` decision procedure. If the iterated instruction accesses a pseudo-register, the process continues to a `stack state change` decision procedure that determines whether the iterated instruction requires a permutation to the register stack (and the associated register stack state). If the `stack state change` decision procedure determines that the iterated instruction requires a stack permutation, the process continues to an `insert register stack permutation instruction` procedure, which inserts instructions to place the register stack in a condition appropriate for the iterated instruction. Next, the register stack state is updated by an `update mapping state` procedure. Thus, the register stack state is responsive to changes in the position of the stack registers in the register stack. Thus, the compiler maintains the state of the register stack responsive to the operation of each instruction that accesses the register stack. See *Wallace* at col. 12, ll. 25 – col. 13, ll. 17.

Appellant submits that none of the above describe references disclose or suggest determining operations to be inserted into the program in order to set each component of the state at each selected program point, wherein the operations assure that the data structure will be in an accurate state at the selected program points. However, the Examiner maintains that *Rosenberg* discloses a process of determining operations to be inserted into the program in order to set each component of the state at each selected program point. For instance, the Examiner asserts:

> Rosenberg teaches storing addresses and variables on the stack, which is done with a push or similar command for storing information into a stack (page 137, lines 28-32). These 'push' commands sets a component, or frame of the stack by pushing information (such as addresses and variables) onto the stack, updating the state of the stack.

(See Final Office Action at page 2, paragraph 8, lines 5-9).

Appellant disagrees with the Examiner's characterization of the *Rosenberg* reference. While *Rosenberg* discloses a procedure call that pushes a return address onto a stack and a child procedure that pushes a parent's frame pointer address onto the stack, there is no mention of the push commands functioning to set a component. Notwithstanding the Examiner's characterization, Appellant submits that *Rosenberg* does not disclose or suggest a process of determining operations to be inserted into the program in order to set each component of the state at each selected program point. Setting a component of a stack and updating the state of the stack for storing information into a stack, as maintained by the Examiner, is not equivalent to determining operations to be inserted into the program in order to set each component of the state at each selected program point.

In addition, the Examiner asserts that *Wallace* teaches "inserting instructions into a program in order to update the state of a stack, hence ensuring that the stack is in a correct state at selected program points. See Final Office Action at page 2, paragraph 8, lines 9-10.

As discussed above *Wallace* does not disclose or suggest determining operations to be inserted into the program in order to set each component of the state at each selected program point, wherein the operations assure that the data structure will be in an accurate state at the selected program points. Particularly, *Wallace* does not disclose a process of determining operations to be inserted into a program in order to set each component of a state at each selected program point. Instead, *Wallace* discloses inserting instructions to place a register stack in a condition appropriate for an iterated instruction. Appellant submits that placing a register stack in a condition appropriate for an iterated instruction is not equivalent to inserting operations into a program to set each component of a state at each selected program point.

Since neither *Rosenberg*, *Wallace* nor *Lo* disclose or suggest determining operations to be inserted into the program in order to set each component of the state at each selected program point, wherein the operations assure that the data structure will be in an accurate state at the selected program points, any combination of *Rosenberg*, *Wallace* and *Lo* would also not disclose or suggest such a feature. Therefore, Claim Group I is patentable over *Rosenberg* in view of *Wallace* and further in view of *Lo*.

For the foregoing reasons, Appellant submits that the Examiner has failed to establish a *prima facie* case of obviousness as set forth in MPEP § 706.02(j). Specifically, the Examiner has failed to show that "[t]he teaching or suggestion to make the claimed combination … [is] found in the prior art, and not based on Appellant's disclosure," as required by In re Vaeck, 947 F.2d 488 (Fed. Cir. 1991).

Claims 2-6 depend from claim 1, claims 11 and 12 depend from claim 11, and claims 14-17 depend from claim 13. Given that dependent claims necessarily include the limitations of the claims from which they depend, Appellant submits that the invention as claimed in claim 2-6, 11, 12 and 14-17 are similarly patentable over *Rosenberg* in view of *Wallace* and further in view of *Lo*.

Thus, the Examiner erred in rejecting claims 1, 2, 10, 13 and 14 under U.S.C. § 103(a).

(B) THE PENDING CLAIMS WERE IMPROPERLY REJECTED UNDER 35 U.S.C. § 103(a) BECAUSE ANY COMBINATION OF *ROSENBERG*, *WALLACE*, *LO*, *GORDON* AND *DUNN* DO NOT DISCLOSE OR SUGGEST DETERMINING A SET OF ONE OR MORE OPERATIONS TO BE INSERTED INTO THE PROGRAM IN ORDER TO SET EACH COMPONENT OF THE STATE AT EACH SELECTED PROGRAM POINT, WHEREIN THE OPERATIONS ASSURE THAT THE DATA STRUCTURE WILL BE IN AN ACCURATE STATE AT THE SELECTED PROGRAM POINTS

Claims 3 and 15 of Claim Group I are not obvious in view of *Rosenberg*, *Wallace*, *Lo* and *Gordon* under 35 U.S.C. § 103(a). *Gordon* discloses a method for building a call stack tree for a software program. See *Gordon* at col. 18, ll. 25-30. Nonetheless, *Gordon* does not disclose or suggest determining a set of one or more operations to be inserted into the program in order to set each component of the state at each selected program point, wherein the operations assure that the data structure will be in an accurate state at the selected program points.

As discussed above, neither *Rosenberg*, *Wallace* nor *Lo* disclose or suggest determining a set of one or more operations to be inserted into the program in order to set each component of the state at each selected program point, wherein the operations assure that the data structure will be in an accurate state at the selected program points. As a result, any combination of *Rosenberg*, *Wallace*, *Lo* and *Gordon* would also not disclose or suggest such a feature.

Claims 4, 5, 11, 12, 16 and 17 of Claim Group I are not obvious in view of *Rosenberg*, *Wallace*, *Lo* and *Dunn* under 35 U.S.C. § 103(a). *Dunn* discloses a translating software emulator designed for converting code from a legacy system to a target system and fully preserving the synchronous exception state while allowing for full and aggressive optimization in the translation. See *Dunn* at Abstract. However, *Dunn* does not disclose or suggest determining a set of one or more operations to be inserted into the program in order to set each component of the state at each selected program point, wherein the operations assure that the data structure will be in an accurate state at the selected program points.

As discussed above, *Rosenberg*, *Wallace* nor Lo disclose or suggest such a feature. Therefore, any combination of *Rosenberg*, *Wallace*, *Lo* and *Dunn* would also not disclose or suggest determining a set of one or more operations to be inserted into the program in order to set each component of the state at each selected program point, wherein the operations assure that the data structure will be in an accurate state at the selected program points.

Claim 6 of Claim Group I is not obvious in view of *Rosenberg*, *Wallace*, *Lo*, *Dunn* and *Gordon* under 35 U.S.C. § 103(a). For the reasons described above, any combination of *Rosenberg*, *Wallace*, *Lo*, *Dunn* and *Gordon* would not disclose or suggest determining a set of one or more operations to be inserted into the program in order to set each component of the state at each selected program point, wherein the operations assure that the data structure will be in an accurate state at the selected program points.

For the foregoing reasons, Appellant submits that the Examiner has failed to establish a *prima facie* case of obviousness as set forth in MPEP § 706.02(j). Specifically, the Examiner has failed to show that "[t]he teaching or suggestion to make the claimed combination ... [is] found in the prior art, and not based on Appellant's disclosure," as required by In re Vaeck, 947 F.2d 488 (Fed. Cir. 1991).

Thus, the Examiner erred in rejecting claims 3-6, 11, 12 and 15-17 under 35 U.S.C. § 103(a) in view of the various combinations of *Rosenberg, Wallace, Lo, Dunn* and *Gordon*.

IX. CONCLUSION

Careful review of the Examiner's rejections shows that the Examiner has failed to provide any reference, or combination of references of the prior art that shows all of the elements of each appealed claim. Therefore, Appellant respectfully submits that all appealed claims in this application are patentable and were improperly rejected by the Examiner during prosecution before the United States Patent and Trademark Office. Appellant respectfully requests that the Board of Patent Appeals and Interferences overrule the Examiner and direct allowance of the rejected claims.

This brief is submitted in triplicate. Applicants respectfully believe that the appeal fee of $330.00 is not required. A check for $330.00 to cover the appeal fee for one other than a small entity as specified in 37 C.F.R. § 1.17(c) was included with our original Appeal Brief filed July 28, 2004. Please charge any shortages and credit any overcharges to our Deposit Account No. 02-2666.

Respectfully submitted,

BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN LLP

Dated: November 9, 2004

Ashley Ott

Ashley R. Ott
Reg. No. 55,515

12400 Wilshire Boulevard
Seventh Floor
Los Angeles, CA. 90025-1026
(408) 720-8598

X. APPENDIX OF CLAIMS (37 C.F.R. § 1.192(c)(9))

The claims on appeal read as follows:

1. For a computer-executable program that operates on a data structure, where the data structure must have a required state at selected program points, a method of transforming said program comprising:

   (A) analyzing the program to determine the state of said data structure at said selected program points;

   (B) partitioning said determined state at each said program point into components that may each be set separately;

   (C) determining operations to be inserted into the program in order to set each component of the state at each selected program point, wherein the operations assure that the data structure will be in an accurate state at the selected program points; and

   (D) placing said operations to eliminate partial redundancies of said operations.

2. The method of claim 1, wherein the data structure stores items on a first-in-last-out basis.

3. The method of claim 2, wherein the states of the data structure are represented as paths on a tree of nodes where:

   (A) each path traverses the tree towards the root; and

   (B) each node on the path represent a component of the state.

4. The method of claim 2, wherein the data structure represents actions to be taken by the program if an exception occurs.

5. The method of claim 4, wherein the selected program points are the points of execution immediately before instructions that might cause an exception.

6. The method of claim 4, further comprising representing the actions to be taken as exception paths in a graph.

10. For a computer-executable program that operates on a data structure, where the data structure must have a required state at selected program points, a method of transforming said program comprising:

(A) analyzing the program to determine the state of an instance of said data structure at said selected program points;

(B) partitioning said instance of said data structure into components;

(C) determining a set of one or more operations to be inserted into the program in order to set each component of the state at each selected program point, wherein the operations assure that the data structure will be in an accurate state at the selected program points;

(D) computing placement of the set of operations to eliminate partial redundancies; and

(E) inserting the set of operations at said program points according to the computed placement.

11. The method of claim 10 wherein the data structure is an exception handling stack.

12. The method of claim 11 wherein the components are a pointer to the exception handling stack and an exception handling data structure.

13. A machine-readable medium having a set of instructions, which when executed by a set of one or more processors, causes said set of processors to perform operations comprising:

(A) analyzing a program that operates on a data structure, which must have a required state at selected program points in the program, to determine the state of an instance of said data structure at said selected program points;

(B) partitioning said instance of said data structure into components;

(C) determining a set of one or more operations to be inserted into the program in order to set each component of the state at each selected program point, wherein the operations assure that the data structure will be in an accurate state at the selected program points;

(D) computing placement of the set of operations to eliminate partial redundancies; and

(E) inserting the set of operations at said program points according to the computed placement.

14. The machine-readable medium of claim 13, wherein the data structure stores items on a first-in-last-out basis.

15. The machine-readable medium of claim 14, wherein the states of the data structure are represented as paths on a tree of nodes where:

(A) each path traverses the tree towards the root; and

(B) each node on the path represent a component of the state.

16. The machine-readable medium of claim 14, wherein the data structure represents actions to be taken by the program if an exception occurs.

17. The machine-readable medium of claim 16, wherein the selected program points are the points of execution immediately before instructions that might cause an exception.